Solving Language Equations and Disequations with Applications to Disunification in Description Logics and Monadic Set Constraints*

Franz Baader[1] and Alexander Okhotin[2]

[1] Institute for Theoretical Computer Science, TU Dresden, Germany
[2] Department of Mathematics, University of Turku, Finland

Abstract. We extend previous results on the complexity of solving language equations with one-sided concatenation and all Boolean operations to the case where also disequations (i.e., negated equations) may occur. To show that solvability of systems of equations and disequations is still in ExpTime, we introduce a new type of automata working on infinite trees, which we call looping automata with colors. As applications of these results, we show new complexity results for disunification in the description logic $\mathcal{FL}_0$ and for monadic set constraints with negation. We believe that looping automata with colors may also turn out to be useful in other applications.

1 Introduction

Equations with formal languages as constant parameters and unknowns are among the basic notions of *formal language theory*, first introduced by Ginsburg and Rice [9], who gave a characterization of the context-free languages by solutions of systems of equations of the resolved form $X_i = \varphi_i(X_1, \ldots, X_n)$. For equations of the general form $\varphi(X_1, \ldots, X_n) = \psi(X_1, \ldots, X_n)$ built using union and two-sided concatenation, testing their solvability is easily shown to be undecidable [15]. The state-of-the-art in this area as of 2007 is presented in a survey by Kunc [11]. More recent work shows that undecidability already holds for equations over a one-letter alphabet with concatenation as the only operation [10,12]. In contrast, solvability of language equations with concatenation restricted to *one-sided concatenation with constants* can often be shown to be decidable by encoding the problem into monadic second-order logic on infinite trees (MSO) [16], but this usually does not yield optimal complexity results.

In *logic for programming and artificial intelligence*, language equations with one-sided concatenation are, for instance, relevant in the context of monadic set constraints and unification in description logics (DLs). *Unification in DLs* has been proposed [4] as a novel inference service that can, for example, be used to detect redundancies in ontologies. As a simple example, assume that one knowledge engineer has defined the concept of "women having only daughters" by the

* Supported by DFG (BA 1122/14-1) and the Academy of Finland (grant 134860).

N. Bjørner and A. Voronkov (Eds.): LPAR-18, LNCS 7180, pp. 107–121, 2012.

concept term Woman ⊓ ∀child.Woman. A second knowledge engineer might represent this notion in a somewhat more fine-grained way, e.g., by using the term Female ⊓ Human in place of Woman. The concept terms Woman ⊓ ∀child.Woman and Female ⊓ Human ⊓ ∀child.(Female ⊓ Human) are not equivalent, but they are meant to represent the same concept. The two terms can obviously be made equivalent by viewing the concept name Woman as a concept variable and replacing it in the first term by the concept term Female ⊓ Human. Unification in DLs checks for the existence of such substitutions, and thus can be used to alert the knowledge engineers to potential redundancies in the ontology. In [4] it was shown that unification in the DL $\mathcal{FL}_0$ can be reduced to *finite solvability*[1] of language equations with one-sided concatenation and union, and that this problem is in turn ExpTime-complete. In [3] it was shown that the same complexity result holds for *solvability*,[2] and in [5] this result was extended to language equations with one-sided concatenation and *all Boolean operations*, and to *other decision problems than just solvability*.

Language equations with one-sided concatenation and all Boolean operations can also be regarded as a particular case of equations on sets of terms, known as *set constraints*, which received significant attention [14] in logic for programming since they can be used in program analysis. In fact, solvability of such language equations corresponds to solvability of *monadic* set constraints, where all function symbols are at most unary. In [1] it was already shown that solvability of monadic set constraints is an ExpTime-complete problem.

In the present paper, we extend the existing results for language equations with one-sided concatenation and all Boolean operations to the case of finite systems of language equations and *disequations* (i.e., negated equations). We will show that solvability and finite solvability of such systems are still *in ExpTime*. The motivation comes again from description logics and from set constraints. Set constraints with negation have been investigated in several papers [8,17,2], where it is shown that solvability in the general case is NExpTime-complete. The exact complexity of the monadic case has, to the best of our knowledge, not been determined yet. In description logics, it makes sense to consider not only unification, but also disunification problems in order to prevent certain unifiers. For example the concept term Woman ⊓ ∀child.Woman also unifies with Male ⊓ Human ⊓ ∀child.(Male ⊓ Human), which could, e.g., be prevented by stating that Woman should not become a subconcept of Male, i.e., that Woman ⊓ Male must not be unified with Woman.

In Section 2, we formally define language equations and disequations with one-sided concatenation and all Boolean operations, and show that their (finite) solvability can be reduced to the existence of certain runs of a corresponding looping tree automaton. In Section 3, we introduce looping tree automata with colors, which can express the condition on the runs formulated in the previous section, and then analyze the complexity of their emptiness problem. Finally, in Section 4 we use these results to determine the complexity of testing (finite)

[1] i.e., existence of a solution consisting of finite languages.

[2] i.e., existence of a solution consisting of arbitrary (not necessarily finite) languages.

solvability of the systems of language (dis)equations introduced in Section 2, and then in turn apply this result to identify the complexity of solving disunification problems in $\mathcal{FL}_0$ as well as monadic set constraints with negation.

2 Language (Dis)equations With One-Sided Concatenation

In this section, we first introduce the language (dis)equations that we want to solve, and then we show how solvability can be reduced to a problem for looping automata working on infinite trees.

2.1 The Problem Definition

Given a finite alphabet Σ and finitely many variables $X_1, \ldots, X_n$, the set of *language expressions* is defined by induction:

- any variable X_i is a language expression;
- the empty word ε is a language expression;
- a concatenation φa of a language expression φ with a symbol $a \in \Sigma$ is a language expression;[3]
- if φ, φ' are language expressions, then so are $(\varphi \cup \varphi')$, $(\varphi \cap \varphi')$ and $(\sim\varphi)$.

Given a mapping $\theta = \{X_1 \mapsto L_1, \ldots, X_n \mapsto L_n\}$ of the variables to languages $L_1, \ldots, L_n$ over Σ, its extension to language expressions is defined as

- $\theta(X_i) := L_i$ for all $i, 1 \leq i \leq n$;
- $\theta(\varepsilon) := \{\varepsilon\}$;
- $\theta(\varphi a) := \theta(\varphi)\cdot\{a\}$ for $a \in \Sigma$;
- $\theta(\varphi \cup \varphi') := \theta(\varphi) \cup \theta(\varphi')$, $\theta(\varphi \cap \varphi') := \theta(\varphi) \cap \theta(\varphi')$, and $\theta(\sim\varphi) := \Sigma^* \setminus \theta(\varphi)$.

We call such a mapping a *substitution*.

A *language equation* is of the form $\varphi = \psi$ and a *language disequation* is of the form $\varphi \neq \psi$, where φ, ψ are language expressions. The substitution θ solves the equation $\varphi = \psi$ (the disequation $\varphi \neq \psi$) iff $\theta(\varphi) = \theta(\psi)$ $(\theta(\varphi) \neq \theta(\psi))$. We are interested in solvability of finite systems of language equations and disequations, where a substitution θ solves such a system iff it solves every (dis)equation in the system. Such a solution is called *finite* iff the languages $L_1 = \theta(X_1), \ldots, L_n = \theta(X_n)$ are finite.

Using the fact that, for any sets M_1, M_2, we have $M_1 = M_2$ iff $(M_1 \setminus M_2) \cup (M_2 \setminus M_1) = \varnothing$ and $M_1 = \varnothing = M_2$ iff $M_1 \cup M_2 = \varnothing$, we can transform a given finite system of language equations and disequations into an equivalent one (i.e., one with the same set of solutions) of the form

$$\varphi = \varnothing, \quad \psi_1 \neq \varnothing, \quad \ldots, \quad \psi_k \neq \varnothing. \tag{1}$$

In order to test such a system for (finite) solvability, we translate it into a looping tree automaton.

[3] Note that the concatenation is *one-sided* in the sense that constants ($a \in \Sigma$) are only concatenated from the right to expressions.

2.2 Translation into Looping Tree Automata

Given a ranked alphabet Γ, where every symbol has a nonzero rank, infinite trees over Γ are defined in the usual way, that is, every node in the tree is labeled with an element $f \in \Gamma$ and has as many successor nodes as is the rank of f. A *looping tree automaton* $\mathcal{A} = (Q, \Gamma, Q_0, \Delta)$ consists of a finite set of states Q, a ranked alphabet Γ, a set of initial states $Q_0 \subseteq Q$, and a transition function $\Delta : Q \times \Gamma \to 2^{Q^*}$ that maps each pair (q, f) to a subset of Q^k, where k is the rank of f. A *run* r of $\mathcal{A}$ on a tree t labels the nodes of t with elements of Q, such that the root is labeled with $q_0 \in Q_0$, and the labels respect the transition function, that is, if a node v has label $t(v)$ in t and label $r(v)$ in r, then the tuple $(q_1, \ldots, q_k)$ labeling the successors of v in r must belong to $\Delta(q, t(v))$. The tree t is *accepted* by $\mathcal{A}$ if there is a run of $\mathcal{A}$ on t. The *language accepted by the looping tree automaton* $\mathcal{A}$ is defined as

$$L(\mathcal{A}) := \{t \mid t \text{ is an infinite tree over } \Gamma \text{ that is accepted by } \mathcal{A}\}.$$

It is well-known that the *non-emptiness problem* for looping tree automata, that is, the question whether, given such an automaton $\mathcal{A}$, the accepted language $L(\mathcal{A})$ is non-empty, is decidable in linear time [7].

When reducing a finite system of language (dis)equations of the form (1) to a looping tree automaton, we actually consider a very restricted case of looping tree automata. Assume that the alphabet used in the system is $\Sigma = \{a_1, \ldots, a_m\}$. Then we restrict our attention to a ranked alphabet Γ containing a single symbol γ of rank m. Thus, there is only one infinite tree, and the labeling of its nodes by γ can basically be ignored. Every node in this tree can be uniquely represented by a word $w \in \Sigma^*$, where each symbol a_i selects the ith successor of a node. Consequently, any run on this tree of a looping tree automaton with set of states Q can be represented as a mapping from Σ^* to Q.

Given a finite system of language (dis)equations of the form (1), let Φ denote the set of all subexpressions of $\varphi, \psi_1, \ldots, \psi_k$. We assume that $\varepsilon, X_1, \ldots, X_n \in \Phi$ (otherwise, we simply add them). In [5] we have shown how to construct a looping tree automaton $\mathcal{A}$ with the set of states $Q := 2^{\Phi}$, and with a 1–1-correspondence between runs of $\mathcal{A}$ and substitutions. To be more precise, given a run $r : \Sigma^* \to Q$ of $\mathcal{A}$, the corresponding substitution $\theta^r = \{X_1 \mapsto L_1^r, \ldots, X_n \mapsto L_n^r\}$ is obtained by defining

$$L_i^r := \{w \in \Sigma^* \mid X_i \in r(w)\}.$$

Conversely, given a substitution $\theta = \{X_1 \mapsto L_1, \ldots, X_n \mapsto L_n\}$, the corresponding run r_θ is

$$r_\theta(w) := \{\xi \in \Phi \mid w \in \theta(\xi)\}.$$

Lemma 1 ([5]). *The mapping of runs to substitutions introduced above is a bijection, and the mapping of substitutions to runs is its inverse.*

How do runs that correspond to solutions look like? Given a substitution θ, the corresponding run r_θ satisfies

$$\xi \in r_\theta(w) \text{ iff } w \in \theta(\xi)$$

for all $\xi \in \Phi$. Recall that our system is of the form (1) and that $\varphi, \psi_1, \ldots, \psi_k$ belong to Φ. Thus, θ solves the equation $\varphi = \varnothing$ iff $\varphi \notin r_\theta(w)$ for all $w \in \Sigma^*$, i.e., the run does not use any states containing φ. Consequently, if we remove from $\mathcal{A}$ all states containing φ, then we obtain an automaton whose runs are in a 1–1-correspondence with the solutions of $\varphi = \varnothing$. Let us call the resulting looping tree automaton $\mathcal{A}_\varphi$. Obviously, the size of $\mathcal{A}_\varphi$ is exponential in the size of the input system of language (dis)equations, and this automaton can be constructed in exponential time. To decide solvability of the equation $\varphi = \varnothing$ it is enough to test whether $\mathcal{A}_\varphi$ has a run, which can be done using the (linear-time) emptiness test for looping tree automata.

However, some of the runs of $\mathcal{A}_\varphi$ may correspond to substitutions that do not solve the disequations. If θ solves the disequation $\psi_i \neq \varnothing$, then there is a $w \in \Sigma^*$ such that $w \in \theta(\psi_i)$, which is equivalent to $\psi_i \in r_\theta(w)$.

Lemma 2. *A run r of $\mathcal{A}_\varphi$ corresponds to a solution of the whole system (1) iff for every $i, 1 \leq i \leq k$, there is a word $w \in \Sigma^*$ such that $\psi_i \in r_\theta(w)$.*

If we view the indices $1, \ldots, k$ as colors and assign to each state q of $\mathcal{A}_\varphi$ the color set $\kappa(q) := \{i \mid \psi_i \in q\}$, then the condition in the lemma can be reformulated as follows: we are looking for runs in which each color occurs in the color set of at least one state. We will show in the next section how one can check whether a run satisfying such an additional "color condition" exists.

Finiteness of a solution can also easily be expressed by a condition on runs. In fact, since we have $w \in \theta(X_i)$ iff $X_i \in r_\theta(w)$, we need to look for runs in which the variables X_i occur only finitely often. Let us call a state q of $\mathcal{A}_\varphi$ a *variable state* if $X_i \in q$ for some $i, 1 \leq i \leq n$.

Lemma 3. *A run r of $\mathcal{A}_\varphi$ corresponds to a* finite *solution of $\varphi = \varnothing$ iff it contains only finitely many variable states, i.e., the set $\{w \in \Sigma^* \mid r(w)$ is a variable state$\}$ is finite.*

3 Looping Tree Automata with Colors

In this section, we first introduce a new type of automata that can express the "color condition" caused by disequations, and then analyze the complexity of the non-emptiness problem for these automata.

Definition 1. *A* looping tree automaton with colors *is of the form $\mathcal{A} = (Q, \Gamma, Q_0, \Delta, K, \kappa)$, where $\mathcal{A} = (Q, \Gamma, Q_0, \Delta)$ is a looping tree automaton, K is a finite set (of colors), and $\kappa : Q \to 2^K$ assigns to every state q a set of colors $\kappa(q) \subseteq K$.*

A run of $\mathcal{A} = (Q, \Gamma, Q_0, \Delta, K, \kappa)$ on a tree t is a run of the underlying looping tree automaton (Q, Γ, Q_0, Δ) on t. The set $\kappa(r)$ of colors of the run r *is defined as*

$$\kappa(r) := \{\nu \in K \mid \text{there is a node } v \text{ in } t \text{ with } \nu \in \kappa(r(v))\}.$$

The run r satisfies the color condition *if $K = \kappa(r)$. The tree t is* accepted by the looping tree automaton with colors $\mathcal{A}$ *if there is a run of $\mathcal{A}$ on t that satisfies the color condition. The* language $L(\mathcal{A})$ accepted by the looping tree automaton with colors $\mathcal{A}$ *is the set of all trees accepted by $\mathcal{A}$.*

3.1 Decidability of the Emptiness Problem

In order to show decidability of the non-emptiness problem for looping tree automata with colors, we reduce it to the non-emptiness problem for Büchi tree automata. A *Büchi tree automaton* $\mathcal{A} = (Q, \Gamma, Q_0, \Delta, F)$ is a looping tree automaton that additionally is equipped with a set F of final states. A run r of this automaton on a tree t satisfies the *Büchi acceptance condition* if, on every infinite path through the tree, infinitely many nodes are labeled with final states. The tree t is *accepted* by the Büchi tree automaton $\mathcal{A}$ if there is a run of $\mathcal{A}$ on t that satisfies the Büchi acceptance condition. Again, the *language* $L(\mathcal{A})$ *accepted by the Büchi tree automaton* $\mathcal{A}$ is the set of all trees accepted by $\mathcal{A}$. It is well-known that the emptiness problem for Büchi tree automata is decidable in quadratic time [18].

Let $\mathcal{A} = (Q, \Gamma, Q_0, \Delta, K, \kappa)$ be a looping tree automaton with colors. The corresponding Büchi tree automaton $\mathcal{B}_\mathcal{A} = (Q', \Gamma, Q'_0, \Delta', F)$ is defined as follows:

- $Q' := Q \times 2^K$;
- $Q'_0 := \{(q, K) \mid q \in Q_0\}$;
- for $q \in Q$, $L \subseteq K$, and $f \in \Gamma$ of arity k we define

$$\Delta'((q, L), f) := \{((q_1, L_1), \ldots, (q_k, L_k)) \mid (q_1, \ldots, q_k) \in \Delta(q, f), L \setminus \kappa(q) \text{ is the union of disjoint sets } L_1, \ldots, L_k\};$$

- $F := Q \times \{\varnothing\}$.

The automaton $\mathcal{B}_\mathcal{A}$ simulates $\mathcal{A}$ in the first components of its states. The second component guesses in which subtree the still required colors are to be found. The Büchi acceptance condition ensures that only runs where these guesses are correct are accepting runs.

Proposition 1. *$L(\mathcal{A}) = L(\mathcal{B}_\mathcal{A})$.*

Proof. First, assume that r is a run of $\mathcal{A}$ on t that satisfies the color condition, i.e., $\kappa(r) = K$. For each color $\nu \in K$, select a node v_ν of t such that $\nu \in \kappa(r(v_\nu))$ and v_ν has minimal distance from the root, i.e., no node u in t strictly above v_ν satisfies $\nu \in \kappa(r(u))$. We now construct a run of $\mathcal{B}_\mathcal{A}$ on t by adding to r the second components of the states of $\mathcal{B}_\mathcal{A}$. Consider an arbitrary node v in t. We assign to this node the color set

$$\lambda(v) := \{\nu \in K \mid v_\nu = v \text{ or } v_\nu \text{ lies below } v\}.$$

The mapping r' from the nodes of t to the states of $\mathcal{B}_\mathcal{A}$ is defined as $r'(v) = (r(v), \lambda(v))$. We claim that this mapping is a run of $\mathcal{B}_\mathcal{A}$ on t that satisfies the Büchi acceptance condition.

To show that r' is indeed a run of $\mathcal{B}_\mathcal{A}$, consider an arbitrary node v of t. Let $v_1, \ldots, v_k$ be the successor nodes of v. We must show that $((r(v), \lambda(v)), t(v)) \rightarrow ((r(v_1), \lambda(v_1)), \ldots, (r(v_k), \lambda(v_k)))$ is a valid transition of $\mathcal{B}_\mathcal{A}$. Since r is a run of $\mathcal{A}$, we have $(r(v_1), \ldots, r(v_k)) \in \Delta(r(v), t(v))$, and thus it is sufficient to show that $\lambda(v) \setminus \kappa(r(v))$ is the disjoint union of $\lambda(v_1), \ldots, \lambda(v_k)$. Pairwise disjointness of the sets $\lambda(v_1), \ldots, \lambda(v_k)$ is an immediate consequence of the fact that we have chosen only one node v_ν for each color ν, and such a node can belong only to one of the successor subtrees of v. To show that

$$\lambda(v) \setminus \kappa(r(v)) = \lambda(v_1) \cup \ldots \cup \lambda(v_k),$$

first observe that $\nu \in \lambda(v_i)$ means that $v_\nu = v_i$ or v_ν lies below v_i. Thus, v_ν lies below v, which shows that $\nu \in \lambda(v)$. Since v_ν was chosen so that it has minimal distance from the root, $\nu \in \kappa(r(v))$ is not possible. Thus, we have shown that $\nu \in \lambda(v_i)$ implies $\nu \in \lambda(v) \setminus \kappa(r(v))$. Conversely, assume that $\nu \in \lambda(v) \setminus \kappa(r(v))$. Then $\nu \in \lambda(v)$ means that $v_\nu = v$ or v_ν lies below v. However, $\nu \notin \kappa(r(v))$ shows that the first option is not possible. Consequently, v_ν belongs to one of the subtrees below v, which yields $\nu \in \lambda(v_i)$ for some $i, 1 \leq i \leq k$.

To show that r' satisfies the Büchi acceptance condition, consider the maximal distance of the color nodes v_ν for $\nu \in K$ from the root. Since K is finite, this maximal distance is a well-defined natural number d. Any node v that has a larger distance from the root than d cannot be equal to or have below itself any of the color nodes. Consequently, $\lambda(v) = \varnothing$. This shows that, in any infinite path in t, infinitely many nodes are labeled by r' with a state of $\mathcal{B}_\mathcal{A}$ whose second component is $\varnothing$. Since these are exactly the final states of $\mathcal{B}_\mathcal{A}$, this shows that r' satisfies the Büchi acceptance condition. Thus, we have shown that any tree accepted by $\mathcal{A}$ is also accepted by $\mathcal{B}_\mathcal{A}$, i.e., $L(\mathcal{A}) \subseteq L(\mathcal{B}_\mathcal{A})$.

To show that the inclusion in the other direction also holds, assume that r' is a run of $\mathcal{B}_\mathcal{A}$ on t that satisfies the Büchi acceptance condition. Let r be the mapping from the nodes of t to Q that is obtained from r' by disregarding the second components of states, i.e., if $r'(v) = (q, L)$, then $r(v) = q$. Obviously, r is a run of $\mathcal{A}$. It remains to show that it satisfies the color condition. Assume that there is a color $\nu \in K$ that does not occur in $\kappa(r)$. We claim that this implies that there is an infinite path in t satisfying the following property: $(*)$ for any node v in this path, the second component of $r'(v)$ contains ν. Since this would imply that r' does not satisfy the Büchi acceptance condition, this then shows that such a color cannot exist, i.e., $K = \kappa(r)$.

To show the existence of an infinite path satisfying property $(*)$, it is sufficient to show the following: if v is a node in t such that the second component L of $r'(v)$ contains ν, then there is a successor node v_i of v such that the second component L_i of $r'(v_i)$ contains ν. The existence of such a successor node is an immediate consequence of the definition of the transition relation of $\mathcal{B}_\mathcal{A}$ and the fact that ν cannot be an element of $\kappa(r(v))$ since we have assumed $\nu \notin \kappa(r)$. □

As an immediate consequence of this proposition we have that the non-emptiness problem for looping tree automata with colors is decidable: given a looping tree automaton with colors $\mathcal{A}$, we can construct $\mathcal{B}_\mathcal{A}$, and then use the quadratic non-emptiness test for Büchi automata. Regarding the complexity of this decision procedure, we can observe that the size of $\mathcal{B}_\mathcal{A}$ is polynomial in the number of states of $\mathcal{A}$, but exponential in the number of colors.

Theorem 1. *The non-emptiness problem for looping tree automata with colors can be decided in time polynomial in the number of states, but exponential in the number of colors.*

The non-emptiness for looping tree automata with colors can actually also be reduced to the one for looping tree automata without colors. However, this reduction is not language-preserving, but only emptiness-preserving. In fact, it is easy to show that looping tree automata with colors are more expressive than looping tree automata (see [6] for proofs of these results).

3.2 The Exact Complexity of the Emptiness Problem

If we consider the complexity of the emptiness test described in the previous subsection w.r.t. the overall size of the input automaton, then the test yields an ExpTime upper bound for the emptiness problem. In this section, we show that the problem is actually NP-complete.

We show *NP-hardness of the non-emptiness problem for looping tree automata with colors* by a simple reduction from SAT, the satisfiability problem for sets of clauses in propositional logic. Let $\mathcal{P} = \{p_1, \ldots, p_n\}$ be a set of propositional variables, and $\mathcal{L} = \mathcal{P} \cup \{\neg p_1, \ldots, \neg p_n\}$ the corresponding set of literals. Recall that a clause c is a set of literals $\{\ell_1, \ldots, \ell_m\}$, which stands for the disjunction $\ell_1 \vee \ldots \vee \ell_m$ of these literals. A set of clauses $\mathcal{C} = \{c_1, \ldots, c_p\}$ is read conjunctively, i.e., a propositional valuation satisfies $\mathcal{C}$ iff it satisfies all clauses in $\mathcal{C}$. Given a set of clauses $\mathcal{C} = \{c_1, \ldots, c_p\}$ built using literals from $\mathcal{L} = \mathcal{P} \cup \{\neg p_1, \ldots, \neg p_n\}$, we define the corresponding looping tree automaton with colors $\mathcal{A}_\mathcal{C} = (Q, \Gamma, Q_0, \Delta, K, \kappa)$ as follows:

- $\Gamma := \{f\}$ where f has arity 1;
- $Q := \mathcal{L} \cup \{q_{\text{loop}}\}$;
- $Q_0 := \{p_1, \neg p_1\}$;
- for $1 \leq i < n$ and $\ell \in \{p_i, \neg p_i\}$ we define $\Delta(\ell, f) := \{p_{i+1}, \neg p_{i+1}\}$;
- for $\ell \in \{p_n, \neg p_n, q_{\text{loop}}\}$ we define $\Delta(\ell, f) := \{q_{\text{loop}}\}$;
- $K := \mathcal{C}$;
- $\kappa(\ell) := \{c \in \mathcal{C} \mid \ell \in c\}$ for $\ell \in \mathcal{L}$ and $\kappa(q_{\text{loop}}) := \varnothing$.

Obviously, the size of $\mathcal{A}_\mathcal{C}$ is polynomial in the size of $\mathcal{L}$ and $\mathcal{C}$.

A run r of $\mathcal{A}_\mathcal{C}$ on the unique infinite tree over Γ contains, for every $i, 1 \leq i \leq n$, either p_i or $\neg p_i$, i.e., it determines a propositional valuation. If this run satisfies the color condition, then every clause c belongs to $\kappa(r)$, i.e., there is a literal ℓ that occurs in r (i.e., ℓ is true in the valuation determined by r) and that is contained in c. This shows that runs satisfying the color condition determine valuations that satisfy all clauses in $\mathcal{C}$. Conversely, a propositional valuation determines a unique run r, by choosing for every i the literal that is true in this valuation. If the valuation satisfies $\mathcal{C}$, then for each clause c one of its literals is true, and thus occurs in r. Consequently, each clause occurs in the color set $\kappa(r)$, which shows that r satisfies the color condition. Therefore, the clause set $\mathcal{C}$ is satisfiable iff $L(\mathcal{A}_\mathcal{C}) \neq \varnothing$.

Since the satisfiability problem for sets of propositional clauses is NP-hard, this shows that the same is true for the non-emptiness problem for looping tree automata with colors.

Proposition 2. *The non-emptiness problem for looping tree automata with colors is NP-hard.*

To show that the *non-emptiness problem for looping tree automata with colors is in NP* we consider the Büchi tree automaton constructed in the previous subsection. But first, we eliminate all states in the given automaton that do not occur in any run: these states can be identified in polynomial time using the emptiness test for looping tree automata [7]. The resulting automaton has the same set of runs on any tree, and thus also accepts the same language.

Let us now assume that all states of the looping tree automaton with colors $\mathcal{A} = (Q, \Gamma, Q_0, \Delta, K, \kappa)$ occur in some run, and that the set of colors K is non-empty.[4] Let $\mathcal{B}_\mathcal{A} = (Q', \Gamma, Q'_0, \Delta', F)$ be the Büchi automaton constructed from $\mathcal{A}$ in the previous section. Call a transition $((q, L), f) \to ((q_1, L_1), \ldots, (q_k, L_k))$ *decreasing* if $|L| > |L_i|$ holds for all $i, 1 \leq i \leq k$. Otherwise, the transition is called *non-decreasing*. The following lemma is an easy consequence of the definition of Δ'.

Lemma 4. *If* $((q, L), f) \to ((q_1, L_1), \ldots, (q_k, L_k))$ *is non-decreasing, then* $\kappa(q) \cap L = \varnothing$ *and there is an* $i, 1 \leq i \leq k$, *such that* $L_i = L$ *and* $L_j = \varnothing$ *for all* $j \neq i$.

Now, assume that r is a run of $\mathcal{B}_\mathcal{A}$ satisfying the Büchi acceptance condition. This run starts with an initial state $(q_0, K) \in Q'_0 = Q_0 \times \{K\}$. If the first transition that is applied is a non-decreasing transition, then there is exactly one successor node n_1 of the root to which r assigns a state with $K \neq \varnothing$ as second component, whereas all the other nodes are assigned states with empty second components (i.e., final states). If another non-decreasing transition is applied to n_1, then there is exactly one successor node of n_1 to which r assigns a state with $K \neq \varnothing$ as second component, etc. Since r satisfies the Büchi acceptance condition, after a finite number of non-decreasing steps we reach a node v to which a decreasing transition is applied. Let this decreasing transition be of the form $((q, K), \_) \to ((q_1, L_1), \ldots, (q_k, L_k))$ (where here and in the following, the alphabet symbol from Γ is irrelevant). Since the transition is decreasing, we have $|K| > |L_i|$ for all $i, 1 \leq i \leq k$. Let $v_1, \ldots, v_k$ be the successor nodes of v, and consider all v_i such that $L_i \neq \varnothing$. We can now apply the same analysis as for the root and K to the nodes v_i and $L_i \neq \varnothing$, i.e., we follow a chain of non-decreasing transitions that reproduce L_i until we find the next decreasing transition. This can be done until all color sets are empty. Basically, this construction yields a finite tree of decreasing transitions satisfying certain easy to check properties (see Definition 2 below). Our NP-algorithm guesses such a tree and checks whether the required properties are satisfied. Before we can formally define the relevant properties of this tree, we need to introduce one more notation.

---

[4] If $K = \varnothing$, then $\mathcal{A}$ is a normal looping tree automaton, for which the non-emptiness problem is decidable in polynomial time.

Let $L \subseteq K$ be a non-empty set of colors and let q, q' be states in Q. We say that q' is *directly L-reachable* from q if there is a transition $(q, \_) \to (q_1, \ldots, q_k)$ in Δ such that $q' = q_i$ for some $i, 1 \leq i \leq k$, and $L \cap \kappa(q) = \varnothing$. Note that this implies that there is a non-decreasing transition $((q, L), \_) \to ((q_1, L_1), \ldots, (q_k, L_k))$ with $L_i = L$ and $L_j = \varnothing$ for $j \neq i$ in the transition relation Δ' of $\mathcal{B}_\mathcal{A}$. We say that q' is *L-reachable* from q if there is a sequence of states $p_0, \ldots, p_\ell$ ($\ell \geq 0$) such that $q = p_0$, $q' = p_\ell$, and p_{i+1} is directly L-reachable from p_i for all $i, 0 \leq i < \ell$.

Definition 2. *Given a looping tree automaton with colors $\mathcal{A}$ and the corresponding Büchi tree automaton $\mathcal{B}_\mathcal{A}$, a* dt-tree *for $\mathcal{B}_\mathcal{A}$ is a finite tree T whose nodes are decreasing transitions of $\mathcal{B}_\mathcal{A}$ such that the following properties are satisfied:*

– *the root of T is of the form $((q, K), \_) \to ((q_1, L_1), \ldots, (q_k, L_k))$ such that q is K-reachable from some initial state of $\mathcal{A}$;*
– *if $((q, L), \_) \to ((q_1, L_1), \ldots, (q_k, L_k))$ is a node in T and $i_1, \ldots, i_\ell$ are all the indices i with $L_i \neq \varnothing$, then this node has ℓ successor nodes of the form $((q'_{i_j}, L_{i_j}), \_) \to \cdots$ such that q'_{i_j} is L_{i_j}-reachable from q_{i_j} for $j = 1, \ldots, \ell$.*

Note that the leaves of a dt-tree are labeled with transitions $((q, L), \_) \to ((q_1, L_1), \ldots, (q_k, L_k))$ for which $L_1 = \ldots = L_k = \varnothing$.

Lemma 5. *We have $L(\mathcal{B}_\mathcal{A}) \neq \varnothing$ iff there exists a dt-tree for $\mathcal{B}_\mathcal{A}$.*

The lemma, whose proof can be found in [6], shows that it is enough to design an algorithm that checks for the existence of a dt-tree. For this to be possible in non-deterministic polynomial time, we need to know that the size of dt-trees is polynomial in the size of $\mathcal{A}$. We can actually show the following linear bound in the number of colors.

Lemma 6. *The number of nodes of a dt-tree is bounded by $2 \cdot |K|$.*

Proof. We call a decreasing transition $((q, L), \_) \to ((q_1, L_1), \ldots, (q_k, L_k))$ *removing* if $L \cap \kappa(q) \neq \varnothing$ and *branching* otherwise. Note that, for a branching transition $((q, L), \_) \to ((q_1, L_1), \ldots, (q_k, L_k))$, there must be indices $i \neq j$ such that L_i and L_j are non-empty.

In a dt-tree, for every color there is exactly one transition removing it, and every removing transition removes at least one color. Consequently, a dt-tree can contain at most $|K|$ removing transitions. Since decreasing transitions that are leaves in a dt-tree are necessarily removing, this also shows that the number of leaves of a dt-tree is bounded by $|K|$.

Any branching transition increases the number of leaves by at least one, which shows that a dt-tree can contain at most $|K| - 1$ branching transitions. Since every decreasing transition is either removing or branching, this completes the proof of the lemma. □

Together with Lemma 5, this lemma yields the desired NP upper bound (see [6] for more details). Given the NP-hardness result of Proposition 2, we thus have determined the exact worst-case complexity of the non-emptiness problem.

Theorem 2. *The non-emptiness problem for looping tree automata with colors is NP-complete.*

4 Applying the Results

We will first show that the results obtained so far allow us to determine the exact complexity of (finite) solvability of finite systems of language (dis)equations with one-sided concatenation.

Proposition 3. *For a given finite system of language (dis)equations of the form (1), solvability and finite solvability are decidable in ExpTime.*

Proof. Let $\mathcal{A}_\phi = (Q, \Gamma, Q_0, \Delta)$ be the looping tree automaton constructed from the system (1) in Section 2.2, and define $K := \{1, \ldots, k\}$ and $\kappa(q) := \{i \in K \mid \psi_i \in q\}$ for all $q \in Q$. According to Lemma 2, the system (1) has a solution iff the looping tree automaton with colors $\mathcal{A} = (Q, \Gamma, Q_0, \Delta, K, \kappa)$ has a run satisfying the color condition, i.e., accepts a non-empty language. As shown in the previous section, from $\mathcal{A}$ we can construct a Büchi automaton $\mathcal{B}_\mathcal{A}$ such that $L(\mathcal{A}) = L(\mathcal{B}_\mathcal{A})$ and the size of $\mathcal{B}_\mathcal{A}$ is polynomial in the number of states, but exponential in the number of colors of $\mathcal{A}$. Since the number of states of $\mathcal{A}$ is exponential in the size of the system (1), but the number of colors is linear in that size, the size of $\mathcal{B}_\mathcal{A}$ is exponential in the size of the system (1). As the emptiness problem for Büchi automata can be solved in polynomial time, this yields the desired ExpTime upper bound for solvability.

For finite solvability, we also must take the condition formulated in Lemma 3 into account, i.e., we are looking for runs of $\mathcal{B}_\mathcal{A}$ such that states of $\mathcal{B}_\mathcal{A}$ whose first components are variable states of $\mathcal{A}$ occur only finitely often. This condition can easily be expressed by modifying the Büchi automaton $\mathcal{B}_\mathcal{A}$, as described in a more general setting in the proof of the next lemma. Since the new Büchi automaton constructed in that proof is linear in the size of the original automaton, this yields the desired ExpTime upper bound for finite solvability. □

Lemma 7. *Let $\mathcal{B} = (Q, \Gamma, Q_0, \Delta, F)$ be a Büchi automaton and $P \subseteq Q$. Then we can construct in linear time a Büchi automaton $\mathcal{B}' = (Q', \Gamma, Q'_0, \Delta', F')$ such that $L(\mathcal{B}') = \{t \mid$ there is a run of $\mathcal{B}$ on t that contains only finitely many states from $P\}$.*

Proof. We define $Q' := Q \times \{1\} \cup (Q \setminus P) \times \{0\}, Q'_0 = Q_0 \times \{1\}, F' := (F \setminus P) \times \{0\}$, and

$$\begin{aligned}
\Delta'((q,1),\gamma) := \{((q_1,i_1),\ldots,(q_k,i_k)) \mid \quad & (q_1,\ldots,q_k) \in \Delta(q,\gamma),\\
& i_j = 1 \text{ if } q_j \in P,\\
& i_j \in \{0,1\} \text{ if } q_j \in Q \setminus P\,\},\\
\Delta'((q,0),\gamma) := \{((q_1,0),\ldots,(q_k,0)) \mid \quad & (q_1,\ldots,q_k) \in \Delta(q,\gamma),\\
& q_1,\ldots,q_k \notin P\,\}.
\end{aligned}$$

Basically, this Büchi automaton guesses (by decreasing the second component of a state to 0) that from now on only states from $Q \setminus P$ will be seen. In fact, once the second component is 0, it stays 0 in all successor states, and only states from $Q \setminus P$ are paired with 0. Since F' contains only states with second component 0, this enforces that on every path eventually only states with second component 0 (and thus first component in $Q \setminus P$) occur. By König's lemma, this implies that a run of $\mathcal{B}'$ satisfying the Büchi acceptance condition contains only finitely many states with second component 1, and thus only finitely many states whose first component belongs to P. □

Since (finite) solvability of language equations that are simpler than the ones considered here are ExpTime-hard [4,3], we thus have determined the exact complexity of (finite) solvability of our systems of language (dis)equations.

Theorem 3. *The problems of deciding solvability and finite solvability of finite systems of language (dis)equations of the form (1) are ExpTime-complete.*

4.1 Disunification in $\mathcal{FL}_0$

Unification in the description logic $\mathcal{FL}_0$ has been investigated in detail in [4]. In particular, it is shown there that solvability of $\mathcal{FL}_0$-unification problems is an ExpTime-complete problem. The ExpTime upper bound is based on a reduction to finite solvability of a restricted form of language equations with one-sided concatenation. In this subsection, we use Theorem 3 to show that this upper bound also holds for $\mathcal{FL}_0$-disunification problems.

Due to the space restriction, we cannot recall syntax and semantics of the description logic (DL) $\mathcal{FL}_0$ and the exact definition of unification in $\mathcal{FL}_0$ here (they can be found in [4] and in [6]). For our purposes, it is enough to recall on an abstract level how such unification problems are translated into language equations. The syntax of $\mathcal{FL}_0$ determines what kind of *concept terms* one can build from given finite sets N_C of concept names and N_R of role names, and the semantics is based on interpretations $\mathcal{I}$, which assign sets $C^\mathcal{I}$ to concept terms C. Two concept terms C, D are equivalent ($C \equiv D$) iff $C^\mathcal{I} = D^\mathcal{I}$ for every interpretation $\mathcal{I}$. An $\mathcal{FL}_0$-unification problem is a finite set of equivalences $C \equiv^? D$, where C, D are $\mathcal{FL}_0$-concept patterns, i.e., $\mathcal{FL}_0$-concept terms with variables. Substitutions replace concept variables by concept terms. A *unifier* σ of a given unification problem is a substitution that solves all its equivalences, i.e., satisfies $\sigma(C) \equiv \sigma(D)$ for all equivalences $C \equiv^? D$ in the problem.

As shown in [4], every unification problem can be transformed in linear time into an equivalent one consisting of a single equation $C_0 \equiv^? D_0$. This equation can then be transformed into a system of language equations, with one language equation $E_{C_0,D_0}(A)$ for every concept name $A \in N_C$.[5] The alphabet of these language equations is the set N_R of role names, and the variables occurring in $E_{C_0,D_0}(A)$ are renamed copies X_A of the variables X occurring in the patterns C_0, D_0. In particular, this implies that the equations $E_{C_0,D_0}(A)$ do not share variables, and thus can be solved independently from each other.

[5] These equations are basically language equations with one-sided concatenation, as introduced in the present paper, but with concatenation of constants from the left rather than from the right. However, one can transform them into equations with concatenation of constants from the right, by reversing all concatenations [4]. We assume from now on that the equations $E_{C_0,D_0}(A)$ are already of this form.

Lemma 8 ([4]). *The equivalence* $C_0 \equiv^? D_0$ *has a unifier iff for all concept names* $A \in N_C$*, the language equations* $E_{C_0,D_0}(A)$ *have finite solutions.*

For disunification, we additionally consider finitely many disequivalences $C_i \not\equiv^? D_i$ for $i = 1, \ldots, k$. A substitution σ solves such a disequivalence iff $\sigma(C_i) \not\equiv \sigma(D_i)$. Disequivalences can now be translated into language disequations $D_{C_i,D_i}(A)$, which are defined like $E_{C_i,D_i}(A)$, with the only difference that equality $=$ is replaced by inequality $\neq$. For a disequivalence it is enough to solve one of the associated language disequations. The following can be shown by a simple adaptation of the proof of Lemma 8 in [4].

Lemma 9. *The disunification problem* $\{C_0 \equiv^? D_0, C_1 \not\equiv^? D_1, \ldots, C_k \not\equiv^? D_k\}$ *has a solution iff for every* $A \in N_C$*, there is a substitution* θ_A *such that*

- $\theta_A(X_A)$ *is finite for all* $A \in N_C$ *and all variables* X *occurring in the problem;*
- θ_A *solves the language equation* $E_{C_0,D_0}(A)$ *for all* $A \in N_C$*;*
- *for every index* $i \in \{1, \ldots, k\}$ *there is a concept name* $A \in N_C$ *such that* θ_A *solves the language disequation* $D_{C_i,D_i}(A)$*.*

In order to take care of the last condition of the lemma, we consider functions $f : \{1, \ldots, k\} \to N_C$. Given such a function f, we define, for each $A \in N_C$, the system of language (dis)equations $DE_f(A)$ as

$$DE_f(A) := \{E_{C_0,D_0}(A)\} \cup \{D_{C_i,D_i}(A) \mid f(i) = A\}.$$

The following theorem is then an immediate consequence of Lemma 9.

Theorem 4. *The disunification problem* $\{C_0 \equiv^? D_0, C_1 \not\equiv^? D_1, \ldots, C_k \not\equiv^? D_k\}$ *has a solution iff there is a function* $f : \{1, \ldots, k\} \to N_C$ *such that, for every concept names* $A \in N_C$*, the system of language (dis)equations* $DE_f(A)$ *has a finite solution.*

Since there are exponentially many functions $f : \{1, \ldots, k\} \to N_C$ and finite solvability of each system of language (dis)equations $DE_f(A)$ can be tested in exponential time by Theorem 3, this yields an overall exponential time complexity. ExpTime-hardness already holds for the special case of unification.

Corollary 1. *Solvability of* $\mathcal{FL}_0$*-disunification problems is ExpTime-complete.*

4.2 Monadic Set Constraints

As already mentioned in [3] and [5], there is a close connection between language equations with one-sided concatenation and monadic set constraints, i.e., set constraints where all function symbols are unary or nullary. For the case of set constraints without negation (i.e., where only inclusions between sets are allowed), it has been known for a long time [1] that the unrestricted case is NExpTime-complete and the monadic one (with at least two unary symbols and at least one nullary symbol) is ExpTime-complete. For the case of set constraints with negation (i.e., where inclusions and negated inclusions between sets are allowed), NExpTime-completeness for the unrestricted case has been shown by several authors [8,17,2], but to the best of our knowledge, the monadic case has not been investigated.

Because of the space constraints, we cannot formally introduce monadic set constraints and their translation into language equations here, but it should be noted that this translation is quite obvious (see [6] for details). In fact, nullary and unary function symbols correspond to the elements of the alphabet and application of unary functions to concatenation. To be more precise, using postfix notation, the term $f_1(f_2(\cdots f_k(a)\cdots))$ can be written as a word $af_k \ldots f_1$. This way, sets of terms can be translated into sets of words, where each word starts with a constant and is followed by a (possibly empty) sequence of unary function symbols. Since they basically have the same syntax rules, positive set constraints can be translated into language equations and negative set constraints into language disequations, so that solutions of the set constraints translate into solutions of the language (dis)equations, as sketched above. In order to translate solutions of the languages (dis)equations back to solutions of the sets constraints, one must make sure that every word occurring in such a solution starts with a constant and is followed by a sequence of unary function symbols. This restriction can easily be enforced by adding appropriate equations. This shows that solvability of finite systems of monadic set constraints with negation can be reduced in polynomial time to solvability of finite systems of language (dis)equations. Since Theorem 3 states an ExpTime upper bound also for solvability, this yields an ExpTime upper bound for solvability of monadic set constraints with negation. ExpTime-hardness already holds for the special case of monadic set constraints without negation [1].

Corollary 2. *Solvability of monadic set constraints with negation is ExpTime-complete.*

5 Conclusion

We have shown that solvability and finite solvability of systems of language (dis)equations are ExpTime-complete, in contrast to their undecidability (Σ_2^0-completeness) in the case of unrestricted concatenation [13]. We have used these results to obtain new complexity results for solving monadic set constraints with negation, and for disunification problems in the DL $\mathcal{FL}_0$. As a tool, we have introduced looping tree automata with colors. Though the results of Section 3 show that a direct reduction to the emptiness problem for Büchi tree automata would be possible, using looping tree automata with colors as intermediate formalism makes the presentation much clearer and easier to comprehend. In addition, we believe that these automata may be of interest also for other applications in logic.

References

1. Aiken, A., Kozen, D., Vardi, M.Y., Wimmers, E.L.: The Complexity of Set Constraints. In: Meinke, K., Börger, E., Gurevich, Y. (eds.) CSL 1993. LNCS, vol. 832, pp. 1–17. Springer, Heidelberg (1994)

2. Aiken, A., Kozen, D., Wimmers, E.L.: Decidability of systems of set constraints with negative constraints. Information and Computation 122(1), 30–44 (1995)
3. Baader, F., Küsters, R.: Unification in a Description Logic with Transitive Closure of Roles. In: Nieuwenhuis, R., Voronkov, A. (eds.) LPAR 2001. LNCS (LNAI), vol. 2250, pp. 217–232. Springer, Heidelberg (2001)
4. Baader, F., Narendran, P.: Unification of concept terms in description logic. Journal of Symbolic Computation 31, 277–305 (2001)
5. Baader, F., Okhotin, A.: On Language Equations with One-sided Concatenation., LTCS-Report LTCS-06-01, Chair for Automata Theory, Institute for Theoretical Computer Science, TU Dresden, A short version has been published in the Proceedings of the 20th International Workshop on Unification, UNIF 2006 (2006), `http://lat.inf.tu-dresden.de/research/reports.html`
6. Baader, F., Okhotin, A.: Solving Language Equations and Disequations Using Looping Tree Automata with Colors, LTCS-Report LTCS-12-01, Chair for Automata Theory, Institute for Theoretical Computer Science, TU Dresden (2012), `http://lat.inf.tu-dresden.de/research/reports.html`
7. Baader, F., Tobies, S.: The Inverse Method Implements the Automata Approach for Modal Satisfiability. In: Goré, R.P., Leitsch, A., Nipkow, T. (eds.) IJCAR 2001. LNCS (LNAI), vol. 2083, pp. 92–106. Springer, Heidelberg (2001)
8. Charatonik, W., Pacholski, L.: Negative set constraints with equality. In: Logic in Computer Science, LICS 1994, Paris, France, pp. 128–136 (1994)
9. Ginsburg, S., Rice, H.G.: Two families of languages related to ALGOL. J. of the ACM 9, 350–371 (1962)
10. Jeż, A., Okhotin, A.: On the Computational Completeness of Equations over Sets of Natural Numbers. In: Aceto, L., Damgård, I., Goldberg, L.A., Halldórsson, M.M., Ingólfsdóttir, A., Walukiewicz, I. (eds.) ICALP 2008, Part II. LNCS, vol. 5126, pp. 63–74. Springer, Heidelberg (2008)
11. Kunc, M.: What Do We Know About Language Equations? In: Harju, T., Karhumäki, J., Lepistö, A. (eds.) DLT 2007. LNCS, vol. 4588, pp. 23–27. Springer, Heidelberg (2007)
12. Lehtinen, T., Okhotin, A.: On Language Equations $XXK = XXL$ and $XM = N$ over a Unary Alphabet. In: Gao, Y., Lu, H., Seki, S., Yu, S. (eds.) DLT 2010. LNCS, vol. 6224, pp. 291–302. Springer, Heidelberg (2010)
13. Okhotin, A.: Strict Language Inequalities and Their Decision Problems. In: Jedrzejowicz, J., Szepietowski, A. (eds.) MFCS 2005. LNCS, vol. 3618, pp. 708–719. Springer, Heidelberg (2005)
14. Pacholski, L., Podelski, A.: Set Constraints: A Pearl in Research on Constraints. In: Smolka, G. (ed.) CP 1997. LNCS, vol. 1330, pp. 549–562. Springer, Heidelberg (1997)
15. Parikh, R., Chandra, A., Halpern, J., Meyer, A.: Equations between regular terms and an application to process logic. SIAM Journal on Computing 14(4), 935–942 (1985)
16. Rabin, M.O.: Decidability of second-order theories and automata on infinite trees. Transactions of the American Mathematical Society 141, 1–35 (1969)
17. Stefánsson, K.: Systems of set constraints with negative constraints are NEXPTIME-complete. In: Logic in Computer Science, LICS 1994, Paris, France, pp. 137–141 (1994)
18. Vardi, M.Y., Wolper, P.: Automata-theoretic techniques for modal logics of programs. Journal of Computer and System Sciences 32, 183–221 (1986)